01

Aplicando o Set no modelo

Transcrição

Vamos continuar com o nosso modelo? Agora que conhecemos um pouco de `Set`, vamos colocar em prática em uma classe nova. Criaremos a classe `Aluno`, com os atributos `nome` e `matricula`, sem esquecer do construtor e dos *getters* (lembre-se de utilizar o atalho do Eclipse):

```
public class Aluno {

    private String nome;
    private int numeroMatricula;

    public Aluno(String nome, int numeroMatricula) {
        this.nome = nome;
        this.numeroMatricula = numeroMatricula;
    }

    public String getNome() {
        return nome;
    }

    public int getNumeroMatricula() {
        return numeroMatricula;
    }
}
```

Para testar, vamos criar uma nova classe também: `TestaCursoComAluno`, porque agora vamos colocar uma coleção de alunos dentro da classe `Curso`.

Para fazer isso, precisaremos decidir qual coleção usar. Pode surgir aqui um debate. Uma `Lista`, por exemplo, pode ser uma boa escolha, porque você quer guardar a ordem, pode existir algum aluno repetido... Porém, estudando nosso domínio, resolvemos ficar com um `Set` de alunos.

Então, na classe `Curso`, vamos adicionar:

```
private Set<Aluno> alunos = new HashSet<>();
```

Vamos começar a testar? Podemos copiar de `TestaCurso2` algumas linhas de código, na qual criamos um curso e adicionamos aulas. Precisamos de alguns alunos também, já que queremos colocá-los num curso. Então:

```
public class TestaCursoComAluno {

    public static void main(String[] args) {

        Curso javaColecoes = new Curso("Dominando as coleções do Java",
                "Paulo Silveira");

        javaColecoes.adiciona(new Aula("Trabalhando com ArrayList", 21));
```

```
        javaColecoes.adiciona(new Aula("Criando uma Aula", 20));
        javaColecoes.adiciona(new Aula("Modelando com coleções", 24));

        Aluno a1 = new Aluno("Rodrigo Turini", 34672);
        Aluno a2 = new Aluno("Guilherme Silveira", 5617);
        Aluno a3 = new Aluno("Mauricio Aniche", 17645);
    }
}
```

Agora podemos fazer uso de algo bem comum no dia a dia da programação Java, que é o TDD (***Test Driven Development***, que traduzido para o português, significa **Desenvolvimento guiado por testes**). Fica muito bom para nós, por exemplo, ter uma maneira de matricular nossos alunos no curso. "Ter uma maneira" significa sempre "criar um método". Então vamos escrevê-lo! Nós podemos fazer isto, mesmo sem ter o método:

```
javaColecoes.matricula(a1);
```

O Eclipse vai nos avisar que esse método não existe, mas isso já sabíamos! Podemos dar `CTRL + 1` nessa linha e selecionar a opção de criá-lo dentro da classe `Curso`. Dentro do método `matricula`, podemos agora deixar o código bem encapsulado, e colocar:

```
public void matricula(Aluno aluno){
    this.alunos.add(aluno);
}
```

Agora o Eclipse deixa de reclamar. Podemos matricular os três alunos no curso:

```
javaColecoes.matricula(a1);
javaColecoes.matricula(a2);
javaColecoes.matricula(a3);
```

Mas para mostrar os alunos do curso, como faremos? Agora é uma boa hora para criarmos o *getter* de alunos em `Curso`. Mas qual é a boa prática? Devolver `Collections.unmodifiableSet()`, o mesma que fizemos com as aulas, assim será impossível que alguém fora da classe possa adicionar ou remover objetos:

```
public Set<Aluno> getAlunos() {
    return Collections.unmodifiableSet(alunos);
}
```

Agora vamos mostrar todos os nossos alunos. Podemos tanto usar o `forEach` comum quanto o especial, com o `lambda` do Java 8, ficando assim:

```
System.out.println("Todos os alunos matriculados: ");
javaColecoes.getAlunos().forEach(aluno -> {
    System.out.println(aluno);
});
```

Vamos rodar e... Apenas objetos! Podemos resolver isso facilmente colocando `aluno.getNome()`, porém, podemos solucionar isso sobrescrevendo o método `toString()` da classe `Aluno`, como já fizemos neste curso:

```
@Override
public String toString() {
    return "[Aluno: " + this.nome + ", matricula: " + this.numeroMatricula + "]";
}
```

Rodando novamente, temos tudo certinho e formatado! Mas perceba que ele não manteve a ordem... Sabemos o porquê disso: O `Set` não garante a ordem dos elementos inseridos. A ordem pode até ser igual com um pouco de sorte e poucos testes, porém não é isso que acontece no mundo real, pois como foi dito, o `Set` não guarda a sequência. Caso você realmente precise de uma ordem fiel, começando necessariamente pelo primeiro elemento inserido, então a melhor opção seria você utilizar uma `List` .

O que aprendemos neste capítulo:

- Aplicação do `Set` no nosso modelo.
- Programação defensiva com conjuntos.